PATENT APPLICATION
EXPRESS MAIL No.EL665602978US
ATTORNEY DOCKET No. CIRC019
CLIENT/MATTER NO. 80142.2009

SYSTEM AND METHOD FOR REFORMATTING DATA TRAFFIC

BACKGROUND OF THE INVENTION

1. Related Applications.

The present invention claims priority from U.S. Provisional Patent Application No. 60/197,490 entitled CONDUCTOR GATEWAY filed on April 17, 2000.

2. Field of the Invention.

The present invention relates, in general, to network information access and, more particularly, to software, systems and methods for reformatting request and response traffic in a data communication system.

3. Relevant Background.

Increasingly, business data processing systems, entertainment systems, and personal communications systems are implemented by computers across networks that are interconnected by internetworks (e.g., the Internet). The Internet is rapidly emerging as the preferred system for distributing and exchanging data. Data exchanges support applications including electronic commerce, broadcast and multicast messaging, videoconferencing, gaming, and the like.

Currently, many Internet services are implemented as client-server systems. The client is typically implemented as a web browser application executing on a network-connected workstation or personal computer,

although mail, news, file transfer and other Internet services are relatively common. The server is typically implemented as a web server at a fixed network address. A client enters a uniform resource locator (URL) or selects a link pointing to a URL where the URL identifies the server and particular content from the server that is desired. The client request traverses the network to be received by the server.

The server then obtains data necessary to compose a response to the client request. For example, the response may comprise a hypertext markup language (HTML) document in a web-based application. HTML and other markup language documents comprise text, graphics, active components, as well as references to files and resources at other servers. In the case of static web pages, the web server may simply retrieve the page from a file system, and send it in an HTTP response packet using conventional TCP/IP protocols and interfaces. In the case of dynamically generated pages, the web server obtains data necessary to generate a responsive page, typically through one or more database accesses. The web server then generates a page, typically a markup language document, that incorporates the retrieved data. Once generated, the web server sends the dynamic page in a manner similar to a static page.

Characteristically, web documents include hyperlinks which comprise pointers that, when selected, cause the user's browser to access additional Internet resources identified by the hyperlink. The references contained within hyperlinks may be absolute (e.g., contain a fully qualified URL to the specified resource) or relative (e.g., contain information that is relative to the current page being viewed). Relative and absolute addressing are,

in theory, functionally equivalent to the client, but, in practice, may cause the systems involved in accessing Internet resources to behave differently. Hence, it is desirable to be able to rewrite links within hypertext documents.

HTML documents, as well as other Internet resources, include a variety of formatting options that affect the manner in which client machines behave. For example, a document may include directly or by reference an image file stored on the server as a bitmap, compressed bitmap, JPEG, or other available image format. However, a client can only display images for which it has installed complementary viewing software. Moreover, there may be physical limitations in the display size, color density, or other feature that prohibit accurate or useful rendering of a particular resource on a particular client.

For many client devices, it is desirable to minimize the code size and, therefore, provide a minimal set of image viewing applications. Client devices may have limited processing power in which case it is undesirable to execute complex image processing and rendering processes on the client device to reformat the network resources into a useable format. These difficulties create significant barriers to deployment of new client device types as the device architecture is constrained to a degree by the format of the data that it is intended to be displayed.

Hence, there is a continuing difficulty in matching the format in which Internet resources exist to the format preferred by the clients that desire access to the Internet resource. While the problem is described in terms of graphics files, it is in fact more pervasive. A particular client may be unable to execute Java, for

example, where the Internet resource includes Java resources. In such cases, it would be desirable to reformat the resource so that it was executable on the client. A need exists for systems and methods for systematically reformatting network resources with little or no effort on the part of site owners in a manner that frees client devices from constraints imposed by the network resource format.

One problem with existing systems is that the web server activities required to generate a page consume significant computing resources within the server. Web servers have limited resources (e.g., buffers, ports, etc.) that must be shared amongst the tasks of maintaining connections, processing requests, accessing data, and rendering pages. Web servers can become overburdened and fail when their limited connection and processing resources are exceeded. The server interface often becomes a critical bottleneck in web site performance. To the extent reformatting resources by a server is even possible, the additional burden placed on the server is undesirable.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a system for delivering network resources involving establishing communication between two computers connected via a network such as request-response traffic between a client and a server. Data contained within the request/response traffic is reformatted at least once in a first intermediary computer between the client and server.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

FIG. 2A shows in block-diagram form significant components of a system in accordance with the present invention;

FIG. 2B shows in block-diagram form significant components of an alternative system in accordance with the present invention;

FIG. 3 shows a domain name system used in an implementation of the present invention;

FIG. 4A shows components of FIG. 2A in greater detail;

FIG. 4B illustrates components of FIG. 2B in greater detail; and

FIG. 5 shows back-end components of FIG. 2B in greater detail.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments,

including wireless network environments, as well as small network environments such as conventional LAN systems.

In general, the present invention provides a set of systems and methods that effect reformatting of data as it is exchanged between two or more network-connected computers such as a client computer and a server. While a client-server connection is a common example, the present invention is applicable to data exchange between peers as well. Moreover, the present invention is described in terms of request-response data exchanges typical in the Internet, but is readily applied to other types of data exchanges such as streaming, broadcast, multicast, and other synchronous and asynchronous message exchange methodologies. In this manner, the server need not be aware of any peculiarities of the client device that may be required to present its resources in a useable fashion on the client. Likewise, the client device need not be aware of or constrained by the limitations of the server or the format of the resources provided by the server.

Essentially, an intermediary server is placed in communication with the client and server to participate in the request/response traffic between the client and server. In this position, the intermediary can be given specific knowledge of the configuration, capabilities and preferred formats of both the clients and servers. In a particular implementation, the intermediary server is implemented by a front-end computer and a back-end computer that are coupled over a network. This enables either or both of the front-end and back-end computers to perform the reformatting functions as needed.

The reformatting functions involve a range of reformatting applied to response messages from a server to a client, including, but not limited to:

- modifying links;
- converting graphics formats;
- changing display size;
- modifying image color depth, brightness, contrast or the colors themselves
- compressing data;
- translating text;
- removing rich content such as images and audio file;
- performing optical character or speech recognition;
- encrypting data;
- compiling/converting active content; and
- converting from HTML to XML.
- converting between a first script format and a second script format (e.g., Java script to ActiveX and vice versa).

Although less frequently needed, reformatting can also be applied to request messages. For example, an HTTP request can be reformatted into a structured query language (SQL) request that can be applied to a database. Further, client requests may include large data sets or parameters that benefit from compression, encryption, and/or reformatting of some type. The present invention is contemplated to encompass all types of reformatting performed in an intermediary server.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented. Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token Ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, or peer-to-peer type network. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another network mechanism such as a fibre channel fabric or conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical

environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Micrsosoft Windows, MacIntosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, EPOC and the like. Moreover, the present invention is readily extended to network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

One feature of the present invention is that front-end servers 201 (shown in Fig. 2B) and/or intermediate servers 206 (shown in Fig. 2A) are implemented as an interchangeable pool of servers, any one of which may be dynamically configured to provide the application services. The embodiments of FIG. 2A and FIG. 2B are not strictly alternative as they may coexist in a network environment. A redirection mechanism, shown in Fig. 3, is enabled to select from an available pool of front-end servers 201 and intermediate servers 206 and direct client request packets from the originating web server to a selected front-end server 201 or intermediary server 206.

In the case of web-based environments, front-end 201, intermediary server 206, and back-end server 203 can be implemented using custom or off-the-shelf web server software. For purposes of this document, a web server is a computer running server software coupled to the World Wide Web (i.e., "the web") that delivers or serves web pages. The web server has a unique IP address and accepts connections in order to service requests by sending back responses. A web server differs from a proxy server or a gateway server in that a web server has resident a set of resources (i.e., software programs, data storage capacity, and/or hardware) that enable it to execute programs to provide an extensible range of functionality such as generating web pages, accessing remote network resources, analyzing contents of packets, reformatting request/response traffic and the like using the resident resources. In contrast, a proxy simply forwards request/response traffic on behalf of a client to resources that reside elsewhere, or obtains resources from a local cache if implemented. A web server in accordance with the present invention may reference external resources of the same or different type as the services

requested by a user, and reformat and augment what is provided by the external resources in its response to the user. Commercially available web server software includes Microsoft Internet Information Server (IIS), Netscape Netsite, Apache, among others.

In the embodiment of Fig. 2A, intermediary servers 206 interact directly with server(s) 210-212. In the embodiment of Fig. 2B, intermediary server 206 is implemented as front-end computer 201 and a back-end computer 203. Front-end server 201 establishes and maintains an enhanced communication channel with a back-end server 203. In either embodiment, intermediary server 206, front-end 201 and/or back-end 203 operate to reformat request and/or response traffic flowing between a client 205 and a server 210-212. The reformatted request (response) is sent on to either server 210-212 or client 205. Preferably, a cache structure such as cache 204 shown in Fig. 2A is used to store reformatted response content so that it can be delivered directly from cache 204 without reference to a server 210-212 to offer improved performance. Alternatively, cache 204 may hold response content before it is reformatted, in which case content used from cache 204 is reformatted after retrieval from cache but before transmission to a requesting client 205.

In the specific examples herein client 205 comprises a network-enabled graphical user interface such as a World Wide Web browser. However, the present invention is readily extended to client software other than conventional World Wide Web browser software. Any client application that can access a standard or proprietary user level protocol for network access is a suitable equivalent. Examples include client applications that act

as front ends for file transfer protocol (FTP) services, voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may serve as a front-end for a network application such as a database management system (DBMS) in which case the client application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client application functions as a front-end to a wireless application protocol (WAP) service or the like.

Fig. 2B illustrates an embodiment in which intermediary server 206 is implemented by cooperative action of a front-end computer 201 and a back-end computer 203. Reformatting processes are performed by front-end 201, back-end 203, or both. As in the embodiment of Fig. 2A, reformatting includes rewriting links, converting graphics formats, altering fonts, font sizes, image sizes, color density, compiling/converting program constructs and the like.

Front-end mechanism 201 serves as an access point for client-side communications. In one example, front-end 201 comprises a computer that sits "close" to clients 205. By "close", "topologically close" and "logically close" it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and servers 210-212. Desirably, front-end computers have as fast a connection as possible to the clients 205. For example, the fastest available connection may be implemented in point of presence (POP)

of an Internet service provider (ISP) 106 used by a particular client 205. However, the placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications it is more practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance between front-end 201 and clients 205 may be desirable in some applications as this distance will allow for selection amongst a greater number front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when particular routes and/or front-ends become congested or otherwise unavailable.

Transport mechanism 202 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from server(s) 210-212. Transport mechanism 202 communicates data packets using a proprietary protocol over the public Internet infrastructure in the particular example. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general-purpose network 101. Unlike the general-purpose Internet, front-end 201 and back-end 203 are programmably assigned to serve accesses to a particular server 210-212 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the server owner. The present invention implements a many-to-many mapping of front-ends to back-ends. Because the front-end to back-end mappings can by dynamically changed, a fixed hardware infrastructure can be logically

reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

A particular advantage of the architectures shown in FIG. 2A and FIG. 2B is that they are readily scaled. In accordance with the present invention, not only can the data itself be distributed, but the functionality and behavior required to reformat content and resources is readily and dynamically ported to any of a number of intermediary computers 206 and/or front-ends 201 and/or back-ends 203. In contrast, conventional web server systems require additional hardware and/or software resources scale. In this manner, any number of client machines 205 may be supported. In a similar manner, a web site owner may choose use multiple servers 210-212 that are co-located or distributed throughout network 101. To avoid congestion, additional front-ends 201 and/or intermediary servers 206 may be implemented or assigned to particular web sites. Each front-end 201 and/or intermediary server 206 is dynamically re-configurable by updating address parameters to serve particular web sites. Client traffic is dynamically directed to available front-ends 201 to provide load balancing.

In the examples, dynamic configuration is implemented by a front-end manager component 207 (shown only in FIG. 2B) that communicates with multiple front-ends 201 and/or intermediary servers 206 to provide administrative and configuration information to front-ends 201. Each front-end 201 includes data structures for storing the configuration information, including information identifying the IP addresses of servers 210-212 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 and/or intermediary servers 206 may include information for

prioritizing particular data, quality of service information and the like.

Similarly, additional back-ends 203 can be assigned to a web site to handle increased traffic. Back-end manager component 209 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold current configuration state, quality of service information and the like. In the particular examples, front-end manager 207 and back-end manager 209 serve multiple servers 210-212 and so are able to manipulate the number of front-ends and back-ends assigned to each server 210 by updating this configuration information. When the congestion for the server 210 subsides, the front-end 201, back-end 203, and/or intermediary server 206 can be reassigned to other, busier servers. These and similar modifications are equivalent to the specific examples illustrated herein.

In order for a client 205 to obtain service from a front-end 201 or intermediate server 206, it must first be directed to a front-end 201 or intermediate server 206 that can provide the desired service. Preferably, client 205 initiates all transactions as if it were contacting the originating server 210. FIG. 3 illustrates a domain name server (DNS) redirection mechanism that illustrates how a client 205 is connected to a front-end 201. The DNS systems is defined in a variety of Internet Engineering Task Force (IETF) documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In a typical environment, a client 205 executes a browser 301, TCP/IP stack 303, and a resolver 305. For reasons of performance and packaging, browser 301, TCP/IP stack 303

and resolver 305 are often grouped together as routines within a single software product.

Browser 301 functions as a graphical user interface to implement user input/output (I/O) through monitor 311 and associated keyboard, mouse, or other user input device (not shown). Browser 301 is usually used as an interface for web-based applications, but may also be used as an interface for other applications such as email and network news, as well as special-purpose applications such as database access, telephony, and the like. Alternatively, a special-purpose user interface may be substituted for the more general-purpose browser 301 to handle a particular application.

TCP/IP stack 303 communicates with browser 301 to convert data between formats suitable for browser 301 and IP format suitable for Internet traffic. TCP/IP stack also implements a TCP protocol that manages transmission of packets between client 205 and an Internet service provider (ISP) or equivalent access point. IP protocol requires that each data packet include, among other things, an IP address identifying a destination node. In current implementations the IP address comprise a 32-bit value that identifies a particular Internet node. Non-IP networks have similar node addressing mechanisms. To provide a more user-friendly addressing system, the Internet implements a system of domain name servers that map alpha-numeric domain names to specific IP addresses. This system enables a name space that is more consistent reference between nodes on the Internet and avoids the need for users to know network identifiers, addresses, routes and similar information in order to make a connection.

The domain name service is implemented as a distributed database managed by domain name servers (DNSs) 307 such as DNS_A, DNS_B and DNS_C shown in FIG. 3. Each DNS relies on <domain name:IP> address mapping data stored in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files typically comprise text files that are read by a local name server, and hence become available through the name servers 307 to users of the domain system.

The user programs (e.g., clients 205) access name servers through standard programs such as resolver 305. Resolver 305 includes an address of a DNS 307 that serves as a primary name server. When presented with a reference to a domain name for a server 210-212, resolver 305 sends a request to the primary DNS (e.g., DNS_A in FIG. 3). The primary DNS 307 returns either the IP address mapped to that domain name, a reference to another DNS 307 which has the mapping information (e.g., DNS_B in FIG. 3), or a partial IP address together with a reference to another DNS that has more IP address information. Any number of DNS-to-DNS references may be required to completely determine the IP address mapping.

In this manner, the resolver 305 becomes aware of the IP address mapping which is supplied to TCP/IP component 303. Client 205 may cache the IP address mapping for future use. TCP/IP component 303 uses the mapping to supply the correct IP address in packets directed to a particular domain name so that reference to the DNS system need only occur once.

In accordance with the present invention, at least one DNS server 307 is owned and controlled by system components of the present invention. When a user accesses

a network resource (e.g., a database), browser 301 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210-212 and server 210-212 performs a redirection of the browser to the system owned DNS server (i.e., DNC_C in FIG. 3). In a second embodiment, domain:address mappings within the DNS system are modified such that resolution of the of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of further redirecting the browser 301 to the best available front-end 201.

Unlike a conventional DNS server, however, the system-owned DNS_C in FIG. 3 receives domain:address mapping information from a redirector component 309. Redirector 309 is in communication with front-end manager 207 and back-end manager 209 to obtain information on current front-end and back-end assignments to a particular server 210-212. A conventional DNS is intended to be updated infrequently by reference to its associated master file. In contrast, the master file associated with DNS_C is dynamically updated by redirector 309 to reflect current assignment of front-end 201 and back-end 203. In operation, a reference to servers 210-212 may result in an IP address returned from DNS_C that points to any selected front-end 201 that is currently assigned to servers 210-212. Likewise, servers 210-212 can identify a currently assigned back-end 203 by direct or indirect reference to DNS_C.

Despite the efficiency of the mechanisms shown in Fig. 3, redirection does take some time and it may be preferable to send subsequent requests for a particular server 210-212 directly to an assigned front-end 201 or intermediary server 206 without redirection. When a web page includes links with absolute references the browser 301 may attempt DNS resolution each time a link is followed. To prevent this, one embodiment of the present invention rewrites these links as a part of its reformatting process. In this manner, even though the server contains only a page with absolute references, the page delivered to a client contains relative references.

FIG. 4A illustrates a first embodiment in which a single intermediary computer 206 is used, whereas FIG. 4B and FIG. 5 illustrate a second embodiment where both front-end 201 and back-end 203 are used to implement the intermediary server 206. In the embodiment of FIG. 4A the intermediary server 206 may be located topologically near the client 205 or servers 210-212 --either alternative provides some advantage and the choice of location is made to meet the needs of a particular application. Like identified components are substantially equivalent in Fig. 4A, Fig. 4B and Fig. 5 and for ease of understanding are not duplicatively described herein. Also, the components shown in Fig. 4A and Fig. 4B are optimized for web-based applications. Appropriate changes to the components and protocols are made to adapt the specific examples to other protocols and data types.

Requests from client 205 are received by a TCP unit 401. TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792,

RFC1112 that are incorporated by reference herein. For ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Preferably, TCP component 401 creates a socket for each request, and returns a received response through the same socket. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement file transfer protocol (FTP), mail protocols such as simple mail transport protocol (SMTP) and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

Where the request data information is amenable to reformatting, it is sent to request reformat unit 408. It is contemplated that client requests in many environments are small, carry little data. In such cases it may be desirable to omit HTTP parser 402 and request reformat unit 408 as the requests may rarely need reformatting. However, other applications may require an HTTP request, for example, to be reformatted to a query language request such as SQL. Also, parser 402 may be useful where intermediary server 206 performs other functions, such as prioritization, encryption, compression, and the like that require request analysis even where request reformatting is not performed.

To improve performance, front-end 201 optionally includes a caching mechanism 403. Cache 403 may be implemented as a passive cache that stores frequently and/or recently accessed web site content or as an active cache that stores web site content that is anticipated to be accessed. Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for data within cache 403. If the request can be satisfied from cache 403 the data is supplied directly without reference to servers 210-212 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by data sever 210-212. Also, cache 403 may proactively update the cache contents even before a request is received for particularly important or frequently used data from servers 210-212. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out, or random eviction. When the requested data is not within cache

403, a request is processed to servers 210-212, and the returned data may be stored in cache 403.

A request for data that is not within cache 403 (or if optional cache 403 is not implemented) will require a reference to server 210-212. Some packets will comprise data that may need to be supplied to server 210-212 (e.g., customer credit information, form data and the like). In these instances, HTTP parser 402 couples to transport component 409

The request or reformatted request is passed to transport component 409 for communication to server 210-212 over channel 411. In a particular embodiment, transport component 409 is implements a TCP/IP layer suitable for transport over the Internet or other IP network. Transport component 409 creates a socket connection for each request that corresponds to the socket created in transport component 401. This arrangement enables responses to be matched to requests that generated the responses. Channel 411 is compatible with an interface to servers 210-212 which may include Ethernet, Fibre channel, or other available physical and transport layer interfaces.

Server 210-212 returns responses to transport component 409 and supplies responses to parser 402. Parser 402 implements similar processes with the HTTP response packets as described hereinbefore with respect to request packets. Significantly, parser 402 identifies the data portion of response packets to allow the data portion to be manipulated by reformat component 406.

Response reformat component 406 examines the data portion of response packets to determine when reformatting is appropriate. Reformatting is appropriate when, for

example, the response comprises an HTML document having absolute references in contained links. In this case, reformat component 406 rewrites the links with relative references. Reformatting may also be appropriate when the response includes a graphic format that cannot be interpreted by the client 205 or may not be appropriate to forward to client 205 due to constrained bandwidth. In such a case, a bitmap file might be converted to a JPEG or GIF file. It is contemplated that reformatting will take place at a wide variety of manners and granularity ranging from reformatting of contained links to substantively reformatting an entire document by changing sizes and layout so that it performs its desired function when presented to a requesting client 205. Component 406 is optionally used to implement data decompression where appropriate, decryption, and handle caching when the returning data is of a cacheable type.

Preferably, intermediary server 206 has some knowledge of client 205's capabilities, needs and preferences in order to make intelligent decisions as to when reformatting is appropriate. This knowledge can be supplied by client 205 as a cookie or parameter in the request itself. Alternatively, this knowledge can be maintained in a user database (not shown) within or accessible to intermediary 206 where the user database associates user identification or network address with particular types of reformatting that are to be performed. Intermediary 206 matches request/response traffic between a particular client 205 and server 210-212 and so can apply the desired reformatting in an intelligent manner based on a particular client's needs.

HTTP component 407 reassembles the response, including any reformatted data, into a format suitable for

use by client 205, which in the particular examples herein comprises a web page transported as an HTTP packet. The HTTP packet is sent to transport component 401 for communication to client 205 on the socket opened when the corresponding request was received. In this manner, from the perspective of client 205, the request has been served by originating server 210-212.

In the embodiment of Fig. 4A and Fig. 5, intermediary server 206 shown in Fig. 2A is implemented by front-end computer 201 and back-end computer 203. A front-end computer 201 refers to a computer located at the client side of network 101 whereas a back-end computer 203 refers to a computer located at the server side of network 101. This arrangement enables reformatting to be performed at either or both computers. Hence, in addition to reformatting data to serve needs of clients 205 and server 210-212, data can be reformatted to improve transport across the communication link 202 coupling front-end 201 and back-end 203. For example, back-end 203 can compress graphics and front-end can apply a corresponding decompression so that client 205 receives the data in substantially the same form as provided by server 210-212, but both client 205 and server 210-212 benefit from improved transport characteristics.

Optionally, front-end 201, back-end 203, and intermediary computer 206 implement security processes, compression processes, encryption processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within any of the functional components (e.g., data blender 404) or implemented as separate functional components within front-end 201, back-end 203 or intermediary 206. Also,

parser 402 may identify priority information transmitted with a request. The prioritization information may be provided by the owners of server 210-212, for example, and may be dynamically altered, statically set, or updated from time to time to meet the needs of a particular application.

In the embodiment of FIG. 4B and FIG. 5, blenders 404 and 504 slice and/or coalesce the data portions of the received packets into more desirable "TMP™ units" that are sized for transport through the TMP mechanism 202. The data portion of TCP packets may range in size depending on client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where compression or other reformatting is applied, the data will vary in size depending on the reformatting processes. Data blender 404 receives information from front-end manager 207 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism 202. Data blenders 404 and 504 also mark the TMP units so that they can be re-assembled at the receiving end.

Data blender 404 may also serve as a buffer for storing packets from all appliances 177 that are associated with front-end 201. In accordance with the present invention, data blender 404 may associate a prioritization value with each packet.

TMP™ mechanisms 405 and 505 implement the transport morphing protocol™ (TMP™) packets used in the system in accordance with the present invention. Transport morphing protocol and TMP are trademarks or registered trademarks of Circadence Corporation in the United States and other countries. Front-end TMP mechanism 405 in cooperation with a corresponding back-end TMP mechanism 505 shown in

FIG. 5 are computer processes that implement the end points or sockets of TMP link 202. The TMP mechanism in accordance with the present invention creates and maintains a stable connection between two processes for high-speed, reliable, adaptable communication.

Another feature of TMP is its ability to channel numerous TCP connections through a single TMP pipe 202. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then combined into a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 202 in order to traffic the TCP connections to their appropriate destinations. TMP includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

An advantage of TMP as compared to traditional protocols is the amount of information about the quality of the connection that a TMP connection conveys from one end to the other of a TMP pipe 202. As often happens in a network environment, each end has a great deal of information about the characteristics of the connection in one direction, but not the other. By knowing about the connection as a whole, TMP can better take advantage of the available bandwidth.

In addition to the reformatting functions described above, reformatting may comprise embedding information within request/response traffic. As noted above, most web pages make reference to one or more files that include text, graphics, or program code such as applets or scripts. When a web page is delivered to a web browser, the browser reads these references, makes new requests to retrieve the referenced information, then renders the

page. This can result in each web page requiring tens of server transactions.

In accordance with the present invention, these references can be resolved by the front-end 201, back-end 203, or intermediary 206 and embedded or in-lined into the response. After reformatting, the page can be delivered to client 205 in a manner that is akin to a static web page in that the rendering browser need not make further requests. However, unlike a static page, the page has actually been dynamically generated according to the requirements of server 210-212. This operation can reduce the number of request/response transactions required to generate a page, especially when some of the referenced data is included in cache 204, front-end cache 403 or back-end cache 503.

Another useful reformatting function involves language translation. Companies that desire to do business internationally often face significant challenges and expense in internationalizing network resources such as web sites. The most common solution is to maintain separate complete copies of the web site in each of a limited number of languages. In accordance with the present invention, reformatting may include automated translation services that convert text files from a first language into a second language. The automated translation can be implemented with any desired degree of accuracy using general-purpose translation processes or special-purpose translation processes specified by the owner of a web site.

In a similar manner, reformatting can change the communication mode of the data from one type to another. For example, a text file can be automatically translated to an audio speech file for delivery to sight impaired

users. This augments reformatting to increase font sizes or change color schemes to make them more readily viewable by sight impaired users. In accordance with the present invention, these services can be offered without any involvement or extra work for the site owner. These services can be offered by the owner of an intermediary computer 206 on a subscription basis, for example, to subscribing users.

Another particular application of the present invention involves using the reformatting mechanisms to accomplish document conversion. There are a number of document standards for various word processing programs including Microsoft Word, Corel Wordperfect, postscript, portable document format, rich text format and the like. Clients must have installed a viewer and/or editor for the appropriate document format. The present invention is readily employed to convert from one document format to another so that the requesting client can use the material as intended on available word processing software.

In a similar manner to document conversion, reformatting in accordance with the present invention may be used to convert a file in a first user-level protocol or language to an alternative user-level protocol or language. For example, an HTML document may be reformatted to an XML or other markup language format, or may be reformatted to wireless application protocol (WAP) for display on WAP devices.

Reformatting may involve consideration of multiple data packets in the request/response stream. Transferring an entire multimedia file or document typically involves multiple response packets. In such cases it is contemplated that the intermediary server 206 may be configured to read and parse all data packets associated

with a particular document, reassemble the document, reformat the document, and then break the formatted document into multiple data packets for transmission.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end supports a DBMS protocol such as SQL. Such implementations not only provide the advantages of the present invention, but also enable a client to access a rich set of network resources with minimal client software.